

本节内容

# 平衡二叉树 (AVL)

关注公众号【研途小时】获取后续课程完整更新！

王道考研/CSKAOYAN.COM

# 知识总览

## 平衡二叉树

定义

插入操作

插入新结点后如何调整“不平衡”问题

查找效率分析

# 平衡二叉树的定义

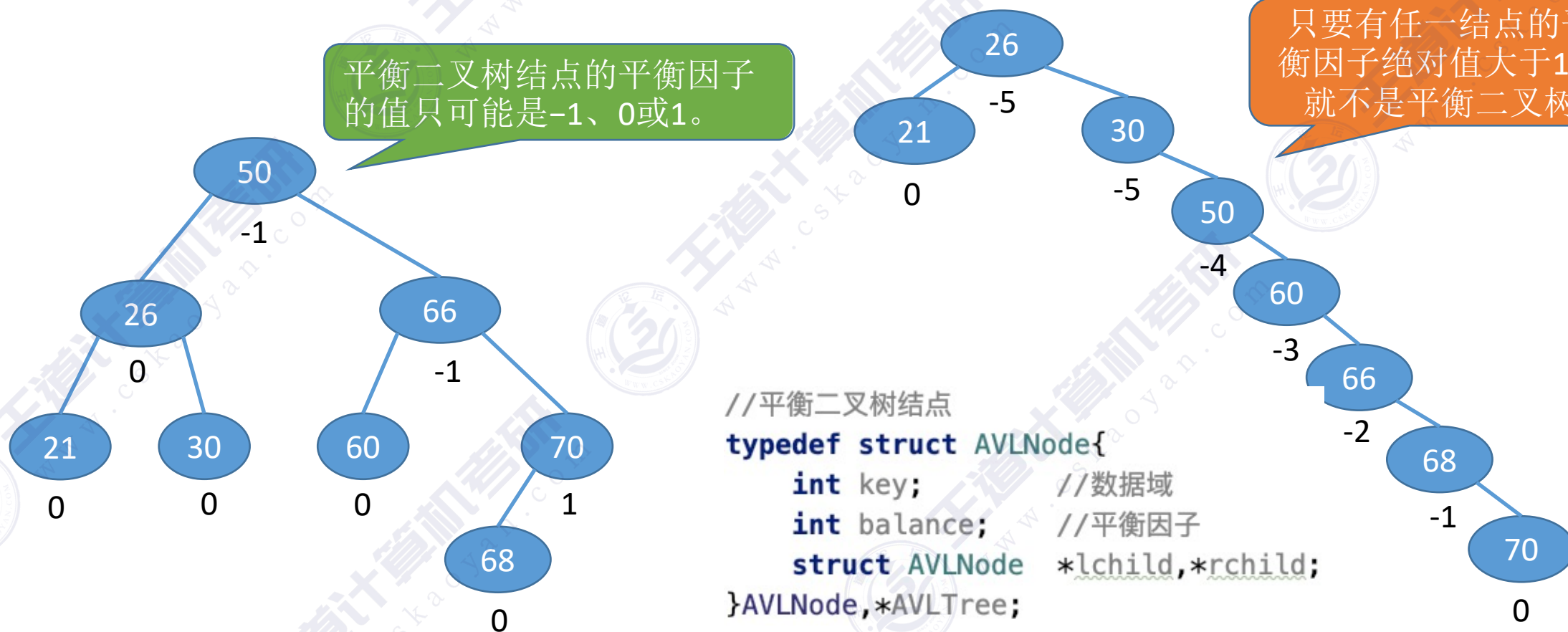
G. M. Adelson-Velsky和  
E. M. Landis

平衡二叉树（Balanced Binary Tree），简称平衡树（AVL树）——树上任一结点的左子树和右子树的高度之差不超过1。

结点的平衡因子=左子树高-右子树高。

平衡二叉树结点的平衡因子的值只可能是-1、0或1。

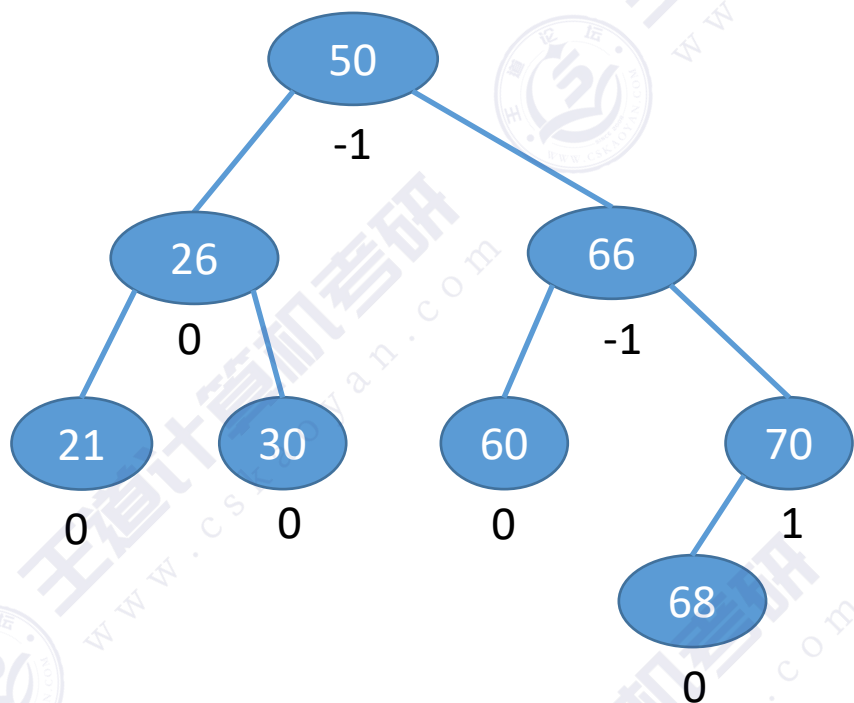
只要有任一结点的平衡因子绝对值大于1，就不是平衡二叉树



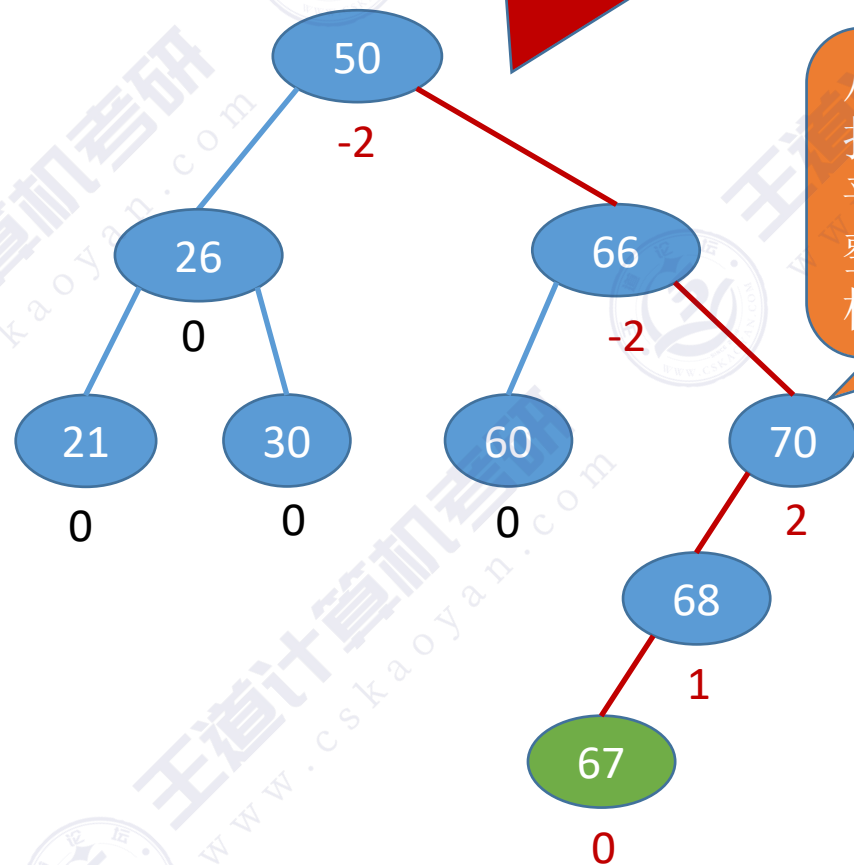
```
//平衡二叉树结点
typedef struct AVLNode{
    int key;           //数据域
    int balance;       //平衡因子
    struct AVLNode *lchild,*rchild;
}AVLNode,*AVLTree;
```

# 平衡二叉树的插入

在二叉排序树中插入新结点后，如何保持平衡？



插入67

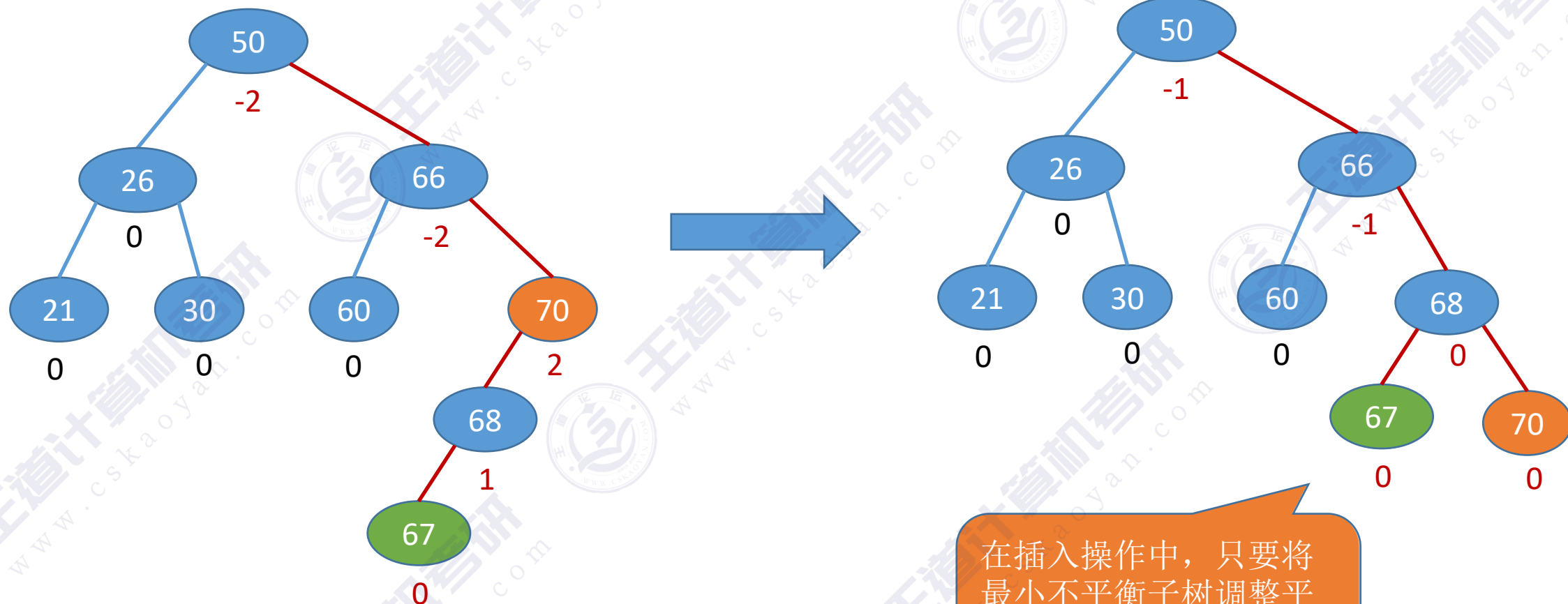


查找路径上的所有结点都有可能受到影响

从插入点往回找到第一个不平衡结点，调整以该结点为根的子树

每次调整的对象都是“最小不平衡子树”

## 平衡二叉树的插入



每次调整的对象都是“最小不平衡子树”

在插入操作中，只要将最小不平衡子树调整平衡，则其他祖先结点都会恢复平衡

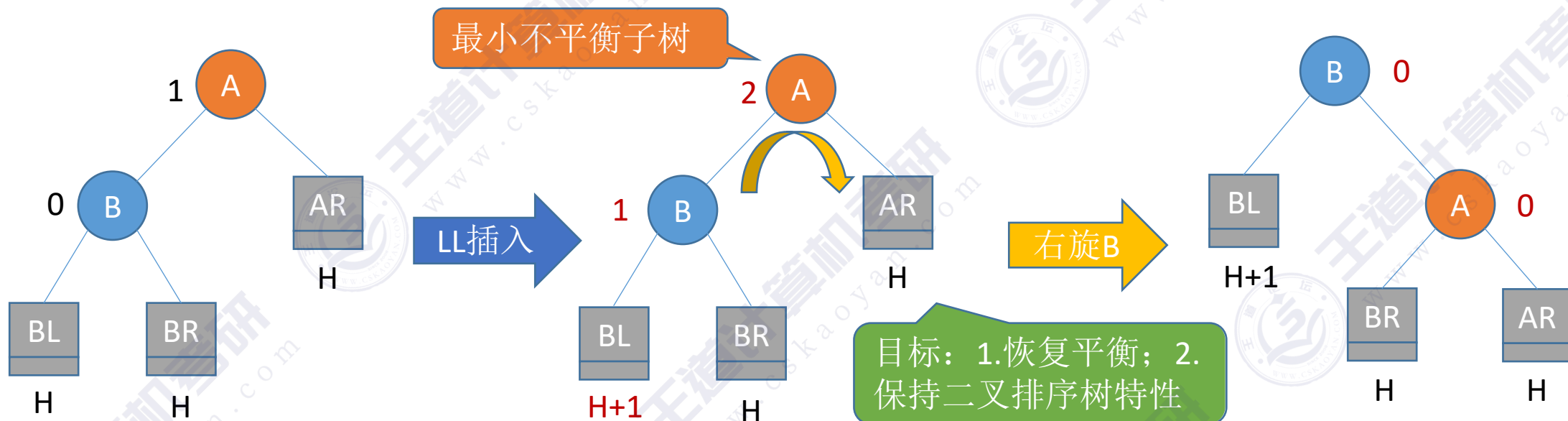
## 调整最小不平衡子树

### 调整最小不平衡子树A

- LL  $\ominus$  在A的左孩子的左子树中插入导致不平衡
- RR  $\ominus$  在A的右孩子的右子树中插入导致不平衡
- LR  $\ominus$  在A的左孩子的右子树中插入导致不平衡
- RL  $\ominus$  在A的右孩子的左子树中插入导致不平衡



## 调整最小不平衡子树 (LL)



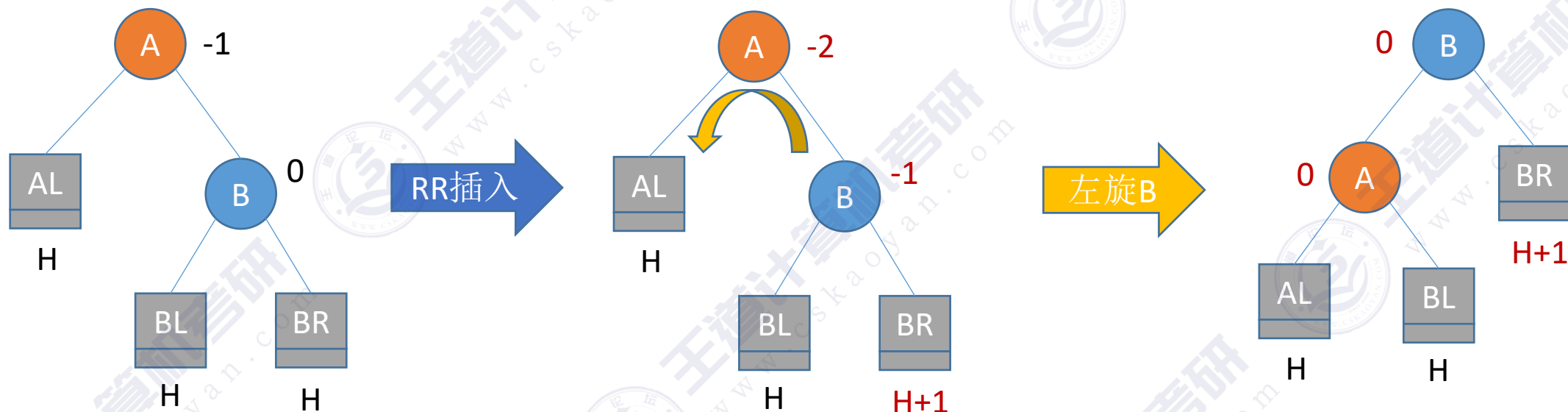
思考: 为什么要假定所有子树的高度都是H?

二叉排序树的特性: 左子树结点值 < 根结点值 < 右子树结点值

$BL < B < BR < A < AR$

1) LL平衡旋转 (右单旋转)。由于在结点A的左孩子 (L) 的左子树 (L) 上插入了新结点, A的平衡因子由1增至2, 导致以A为根的子树失去平衡, 需要一次向右的旋转操作。将A的左孩子B向右上旋转代替A成为根结点, 将A结点向右下旋转成为B的右子树的根结点, 而B的原右子树则作为A结点的左子树。

## 调整最小不平衡子树 (RR)



二叉排序树的特性：左子树结点值 < 根结点值 < 右子树结点值

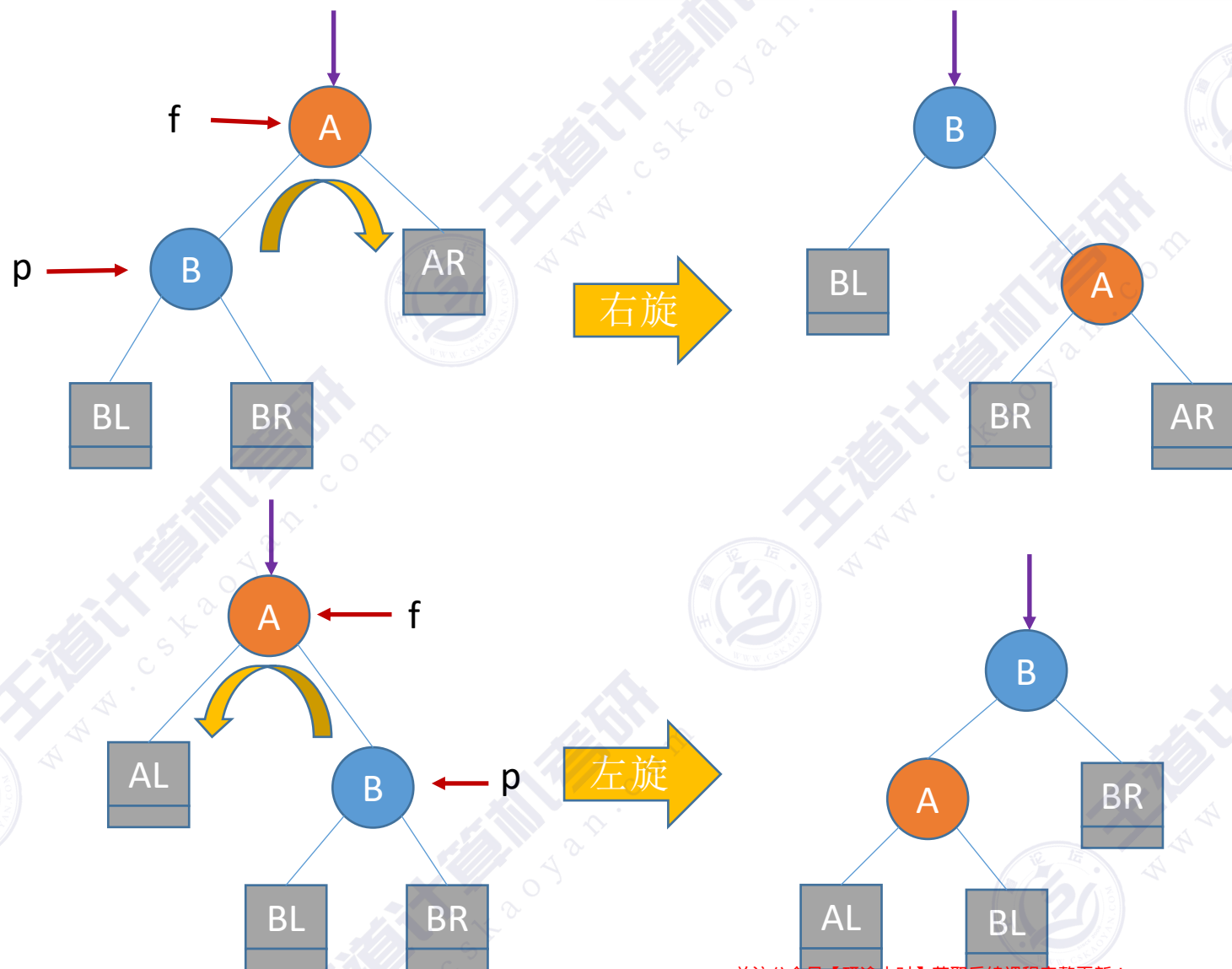
$$AL < A < BL < B < BR$$

2) RR平衡旋转 (左单旋转)。由于在结点A的右孩子 (R) 的右子树 (R) 上插入了新结点, A的平衡因子由-1减至-2, 导致以A为根的子树失去平衡, 需要一次向左的旋转操作。将A的右孩子B向左上旋转代替A成为根结点, 将A结点向左下旋转成为B的左子树的根结点, 而B的原左子树则作为A结点的右子树



## 代码思路

gf->lchild/rchild



实现 **f 向右下旋转**, **p 向右上旋转**:  
其中 f 是爹, p 为左孩子, gf 为 f 他爹

- ①  $f \rightarrow lchild = p \rightarrow rchild;$
- ②  $p \rightarrow rchild = f;$
- ③  $gf \rightarrow lchild/rchild = p;$

**$BL < B < BR < A < AR$**

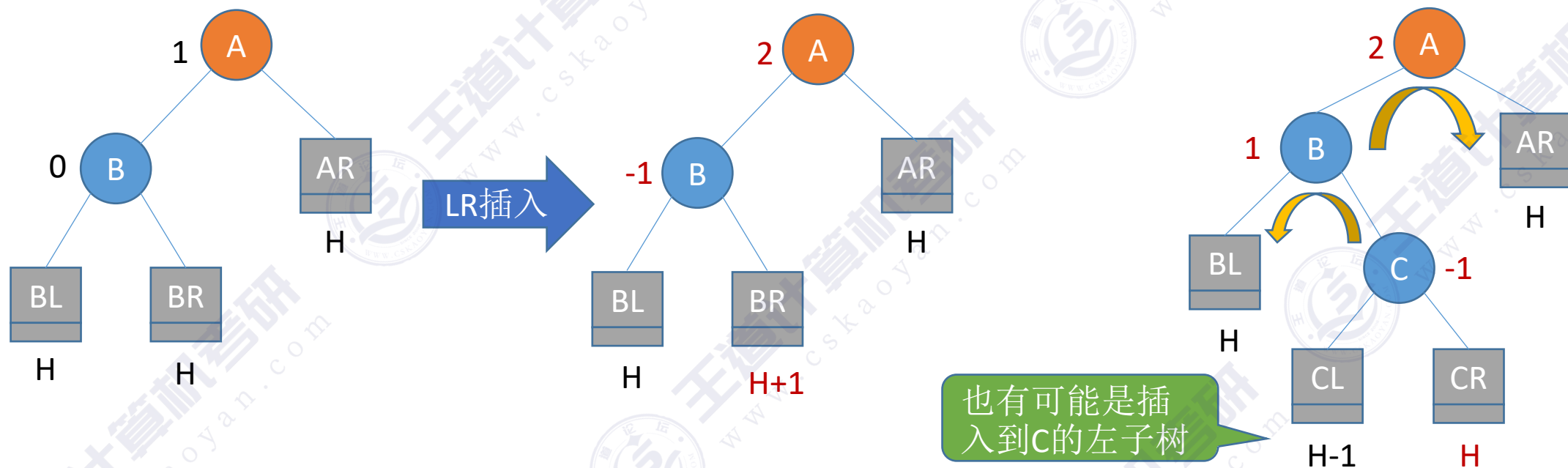
左旋、右旋操作后可以  
保持二叉排序树的特性

实现 **f 向左下旋转**, **p 向左上旋转**:  
其中 f 是爹, p 为右孩子, gf 为 f 他爹

- ①  $f \rightarrow rchild = p \rightarrow lchild;$
- ②  $p \rightarrow lchild = f;$
- ③  $gf \rightarrow lchild/rchild = p;$

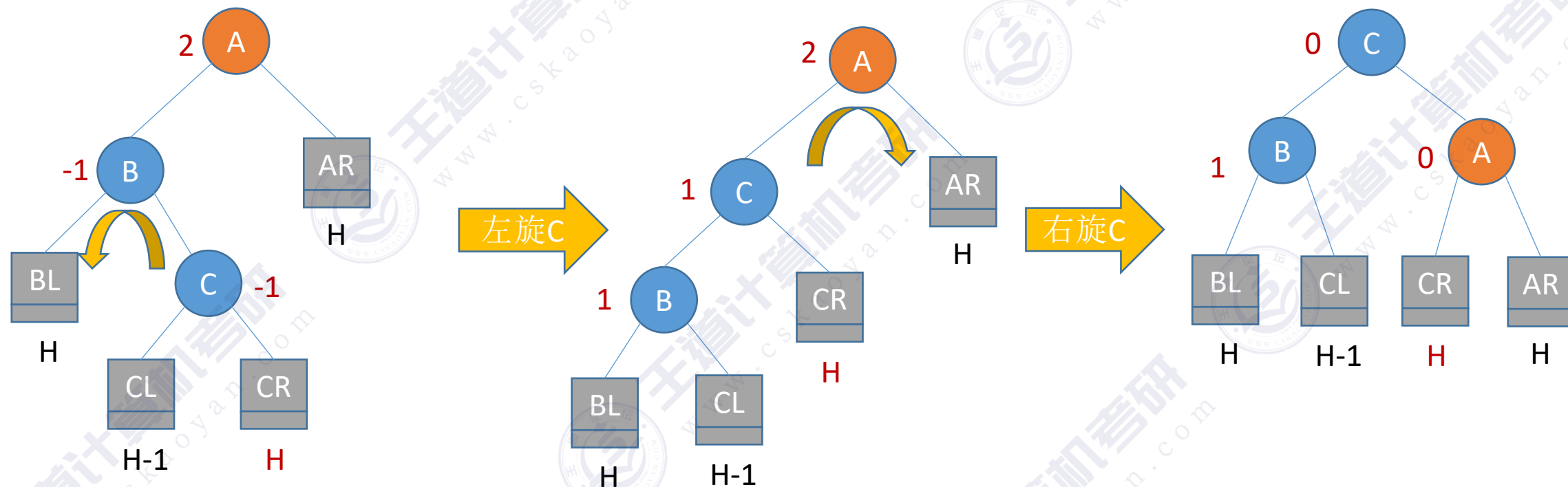
**$AL < A < BL < B < BR$**

## 调整最小不平衡子树 (LR)



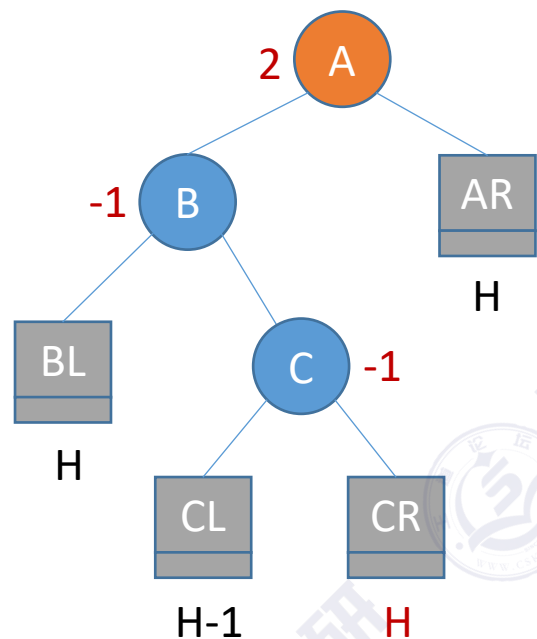
3) LR平衡旋转（先左后右双旋转）。由于在A的左孩子（L）的右子树（R）上插入新结点，A的平衡因子由1增至2，导致以A为根的子树失去平衡，需要进行两次旋转操作，先左旋转后右旋转。先将A结点的左孩子B的右子树的根结点C向左上旋转提升到B结点的位置，然后再把该C结点向右上旋转提升到A结点的位置

## 调整最小不平衡子树 (LR)

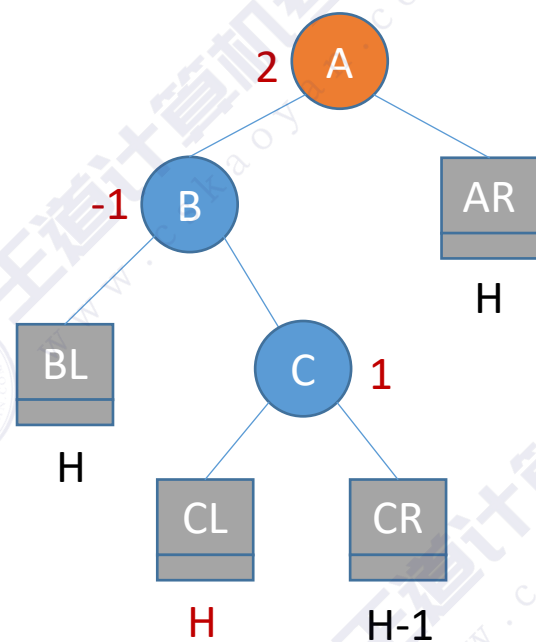
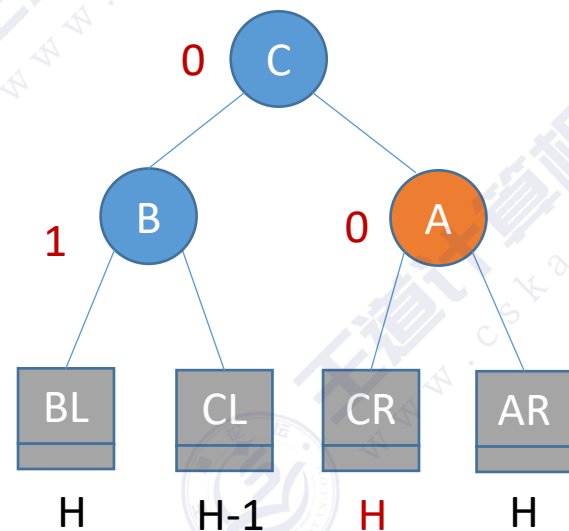


$BL < B < CL < C < CR < A < AR$

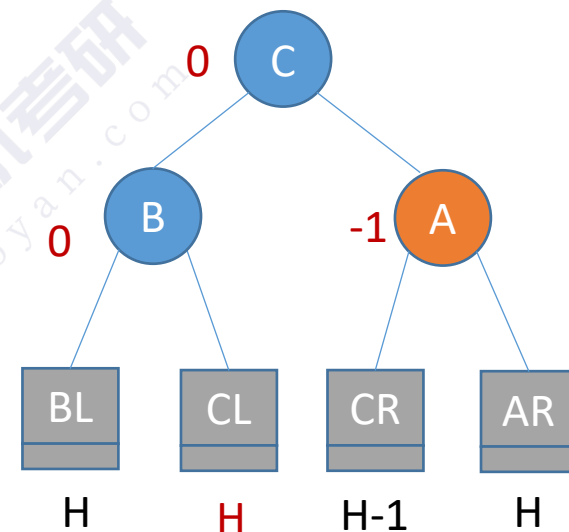
## 调整最小不平衡子树 (LR)



左旋C+右旋C

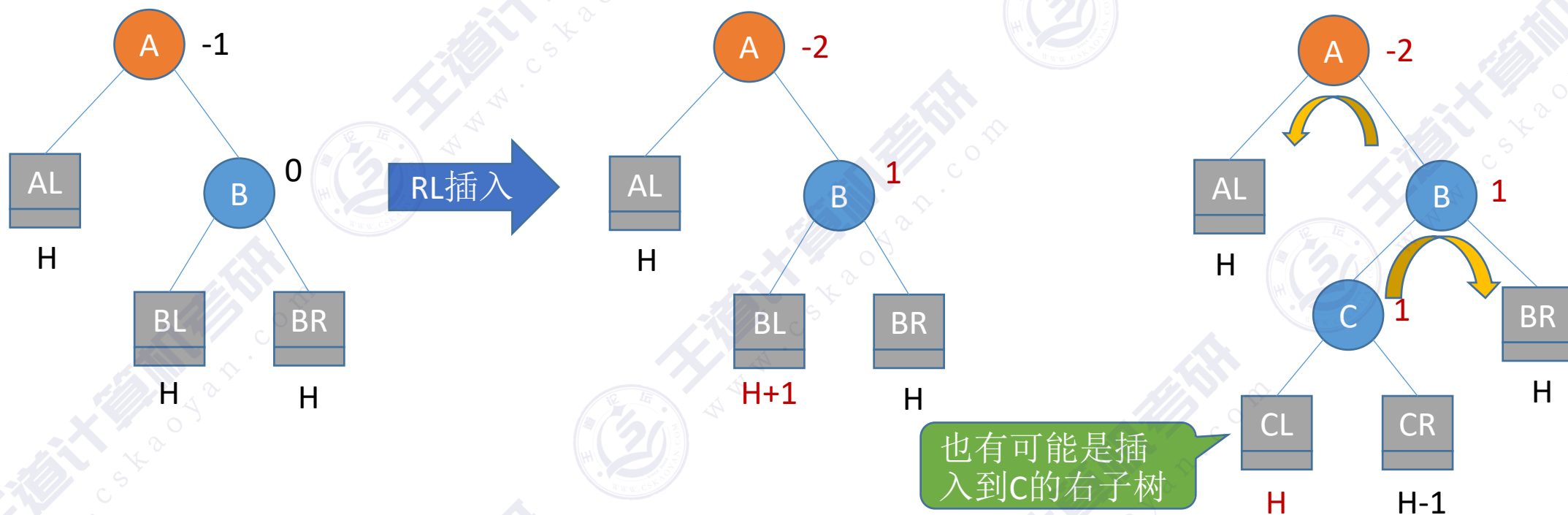


左旋C+右旋C



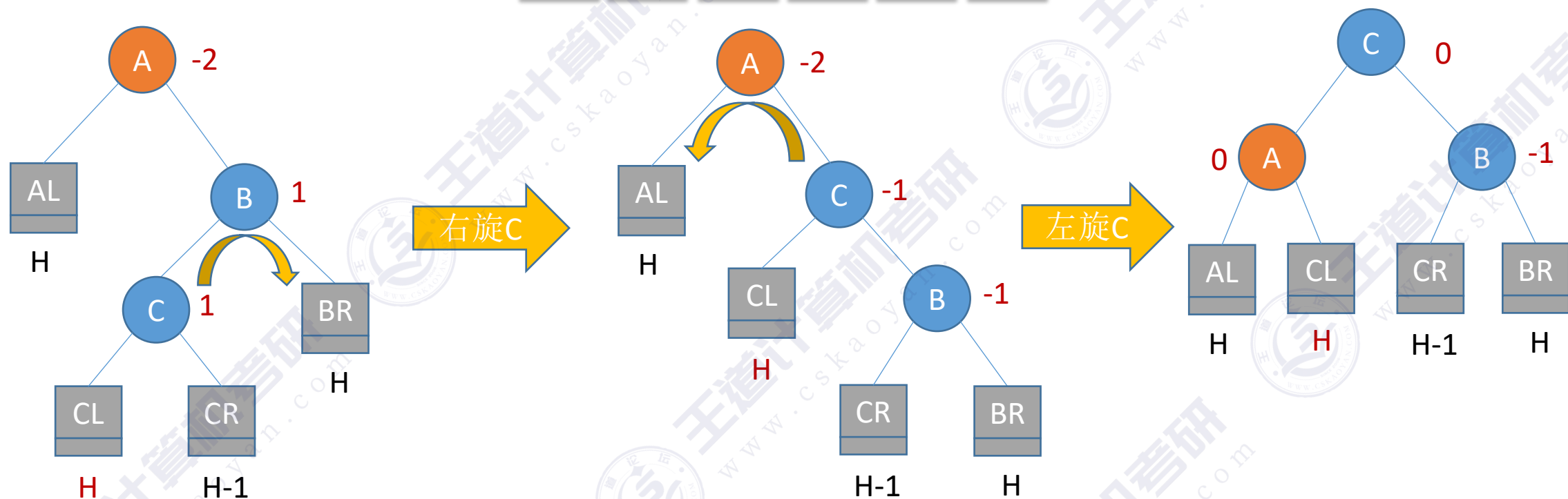
## 调整最小不平衡子树 (RL)

AL < A < CL < C < CR < B < BR



4) RL平衡旋转 (先右后左双旋转)。由于在A的右孩子 (R) 的左子树 (L) 上插入新结点, A的平衡因子由-1减至-2, 导致以A为根的子树失去平衡, 需要进行两次旋转操作, 先右旋转后左旋转。先将A结点的右孩子B的左子树的根结点C向右上旋转提升到B结点的位置, 然后再把该C结点向左上旋转提升到A结点的位置

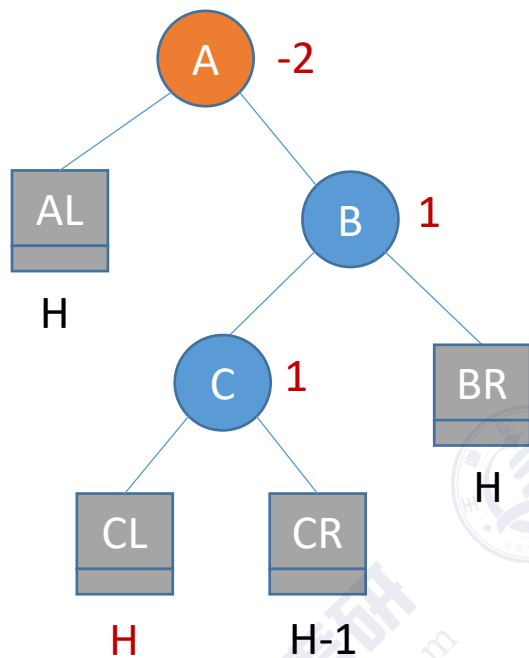
## 调整最小不平衡子树 (RL)



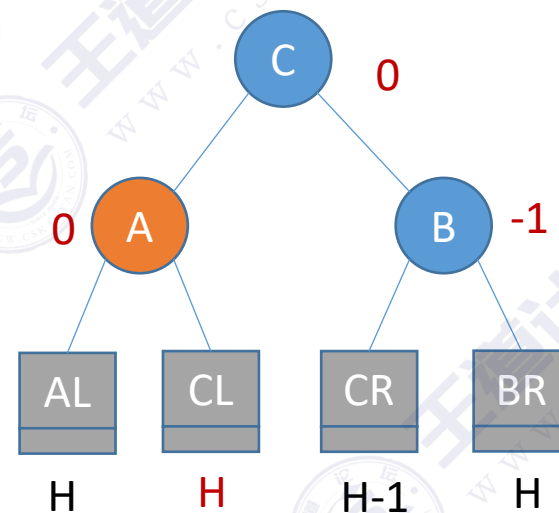
AL < A < CL < C < CR < B < BR



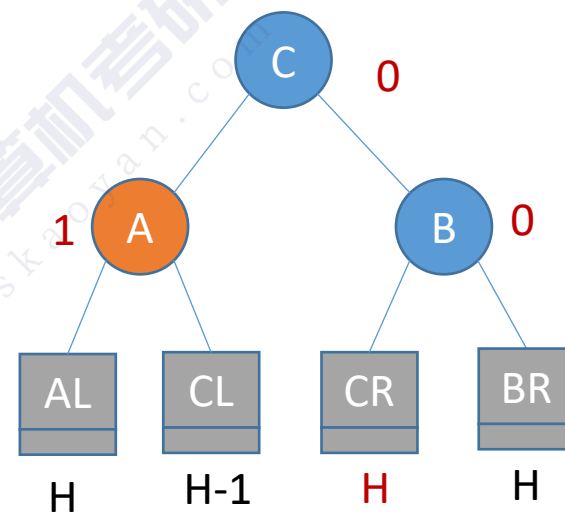
## 调整最小不平衡子树 (RL)



右旋C+左旋C



右旋C+左旋C



## 调整最小不平衡子树

只有左孩子  
才能右上旋

实现 f 向右下旋转，p 向右上旋转：  
其中 f 是爹，p 为左孩子，gf 为 f 他爹

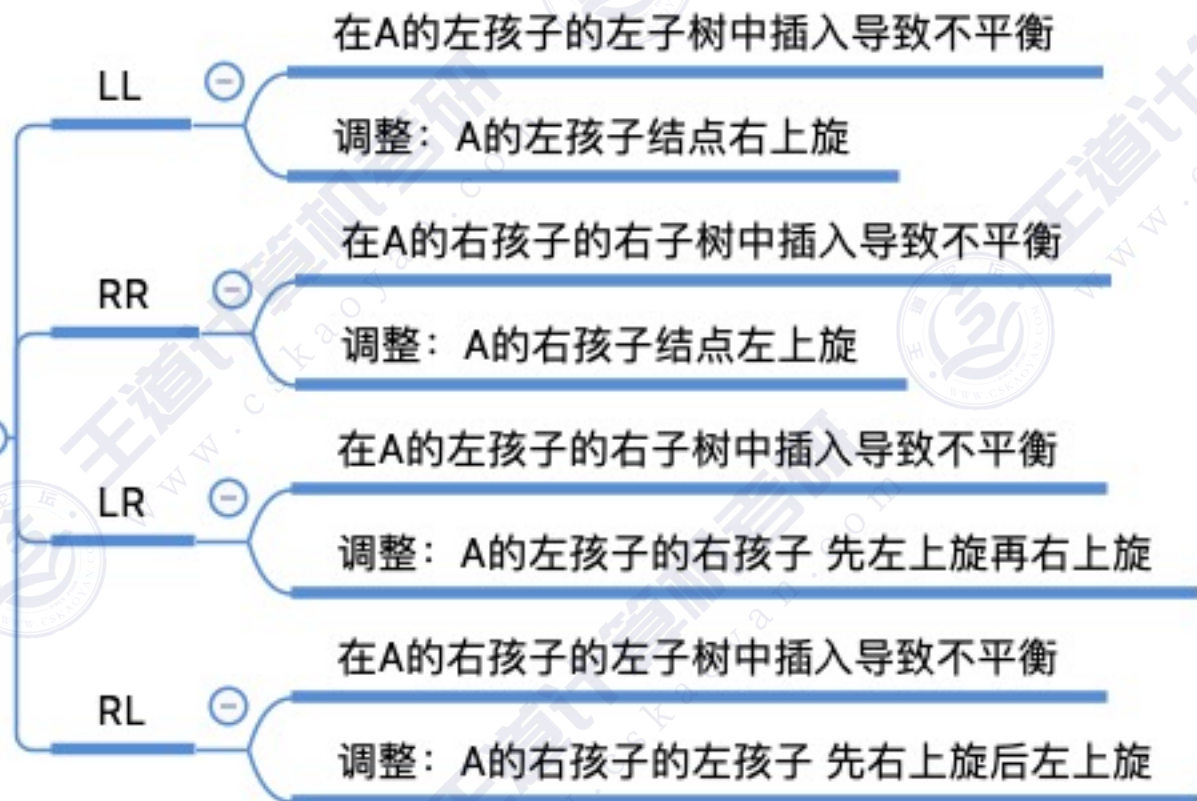
- ①  $f \rightarrow lchild = p \rightarrow rchild;$
- ②  $p \rightarrow rchild = f;$
- ③  $gf \rightarrow lchild/rchild = p;$

### 调整最小不平衡子树A

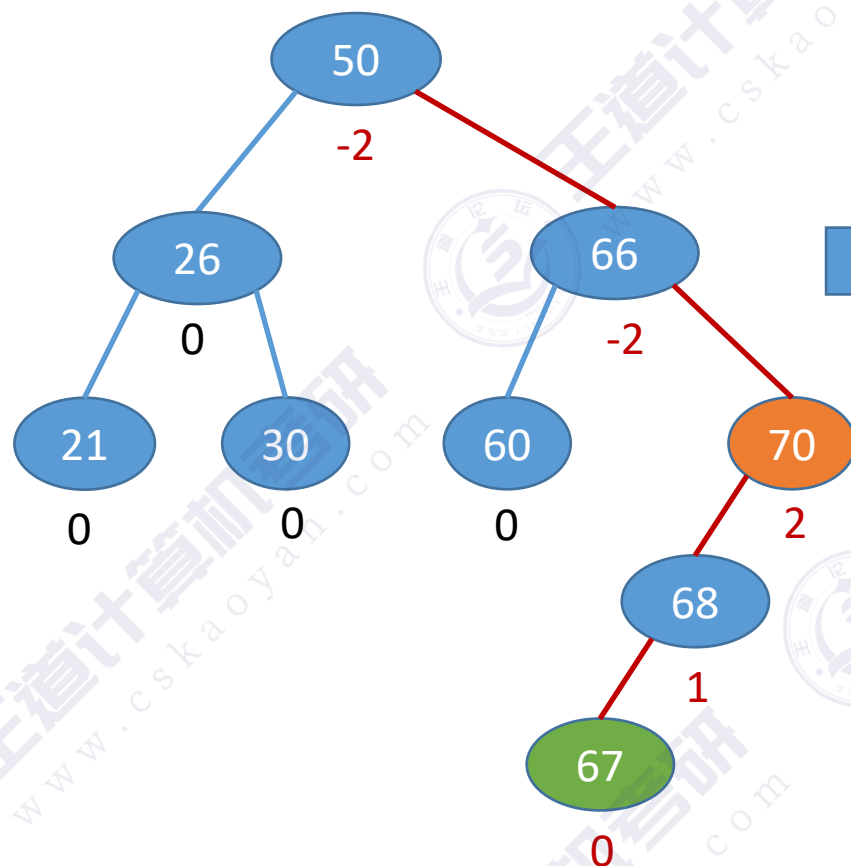
实现 f 向左下旋转，p 向左上旋转：  
其中 f 是爹，p 为右孩子，gf 为 f 他爹

- ①  $f \rightarrow rchild = p \rightarrow lchild;$
- ②  $p \rightarrow lchild = f;$
- ③  $gf \rightarrow lchild/rchild = p;$

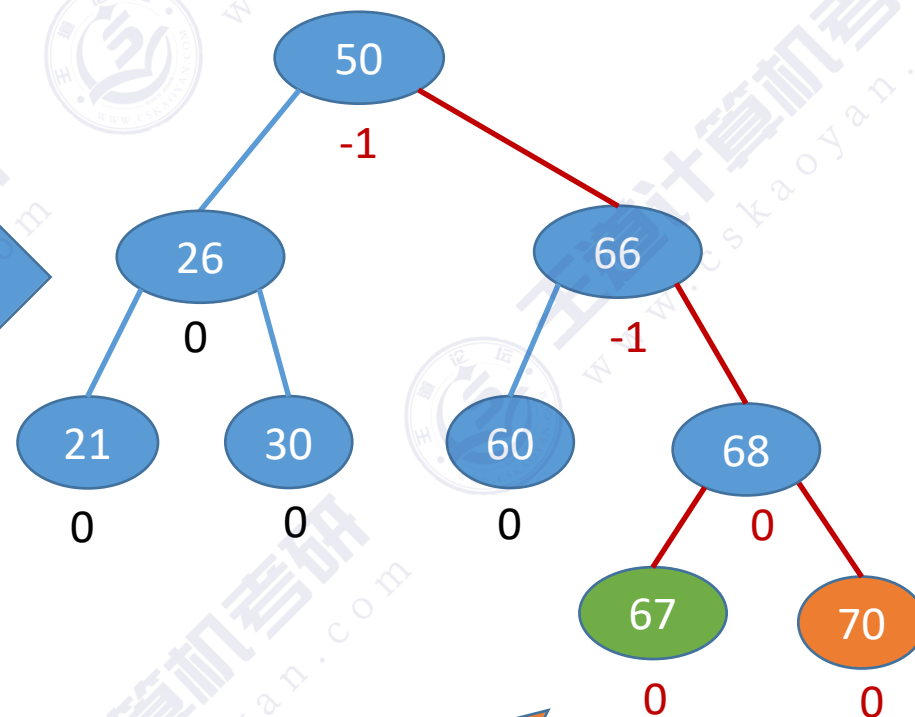
只有右孩子  
才能左上旋



## 填个坑



LL型，左孩子右旋



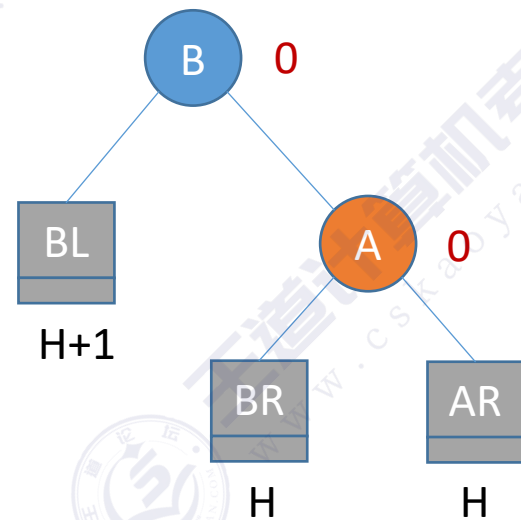
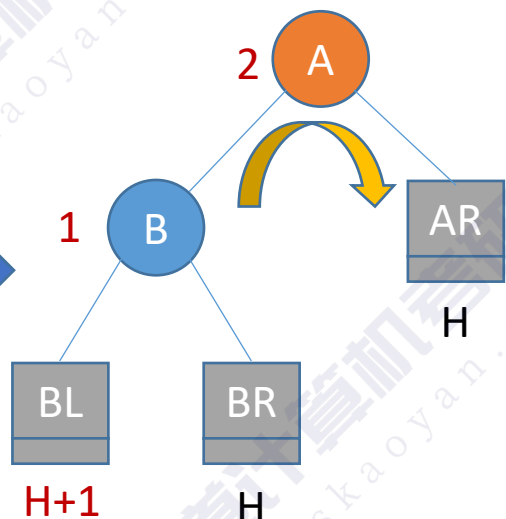
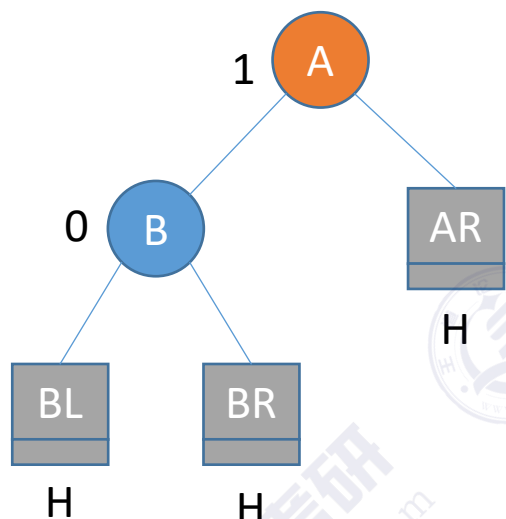
每次调整的对象都是“最小不平衡子树”



WHY?

在插入操作中，只要将最小不平衡子树调整平衡，则其他祖先结点都会恢复平衡

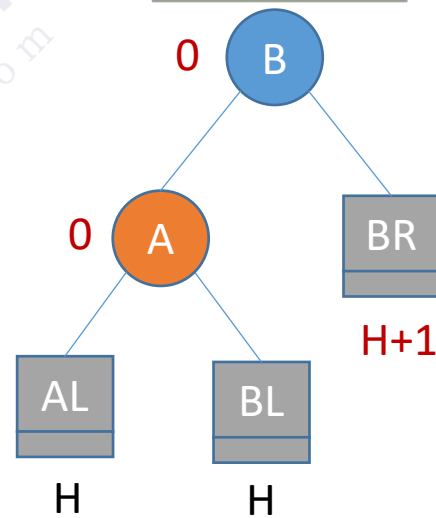
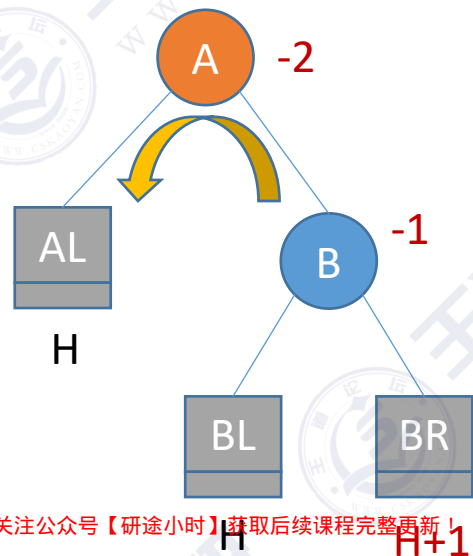
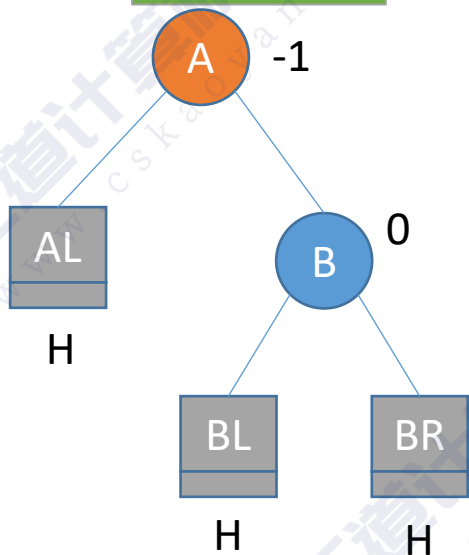
# 填个坑



树高H+2

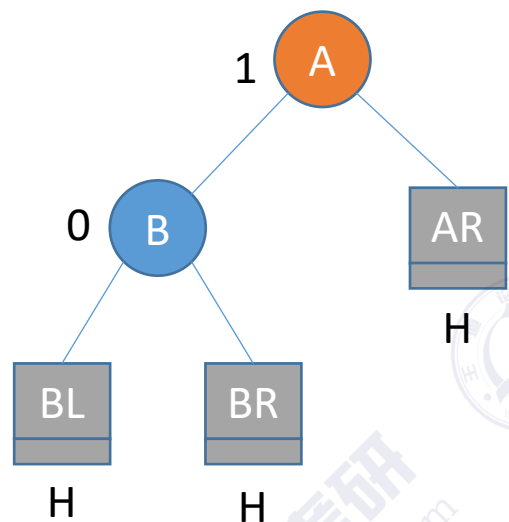
树高H+3

树高H+2



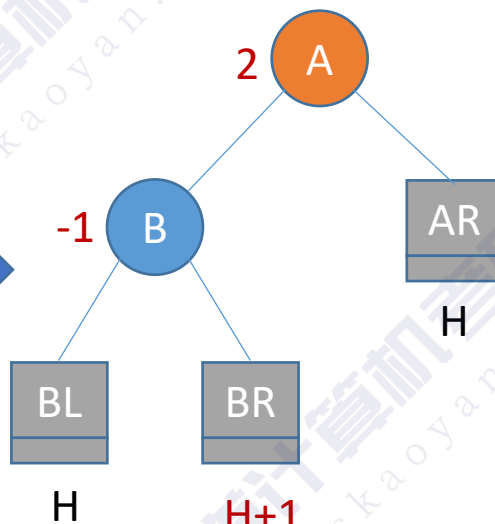
关注公众号【研途小时】获取后续课程完整更新！

# 填个坑



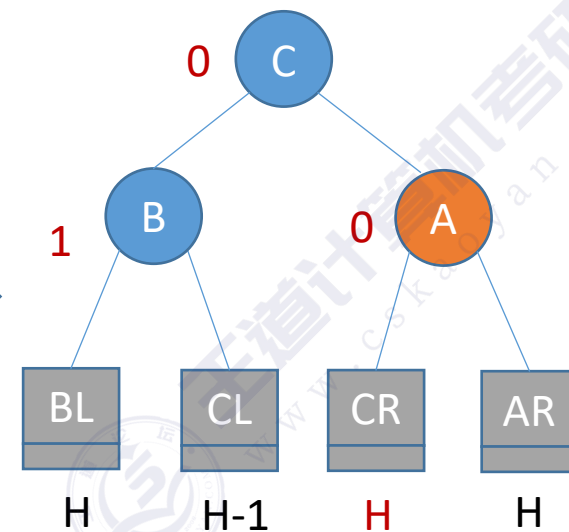
树高H+2

LR型

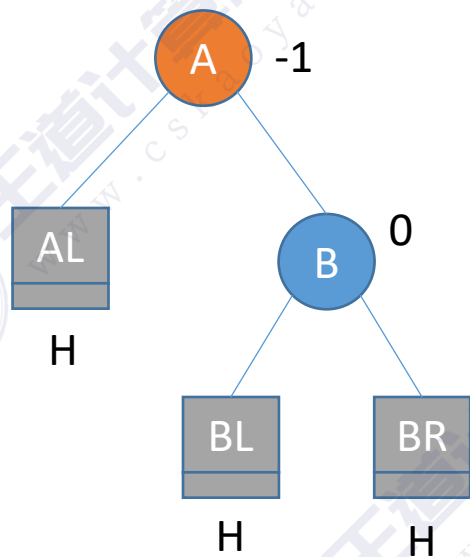


树高H+3

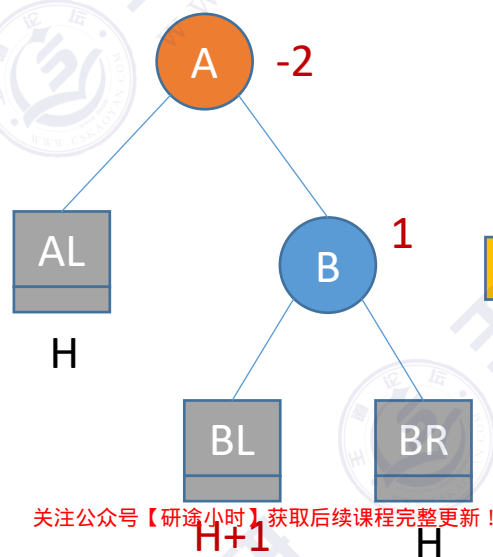
左旋C+右旋C



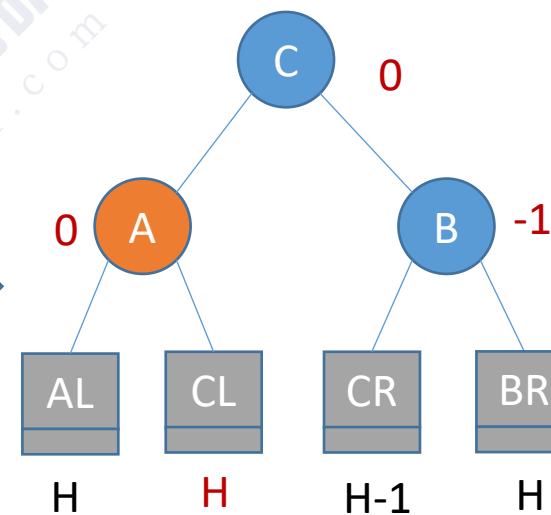
树高H+2



RL型



右旋C+左旋C



关注公众号【研途小时】获取后续课程完整更新！

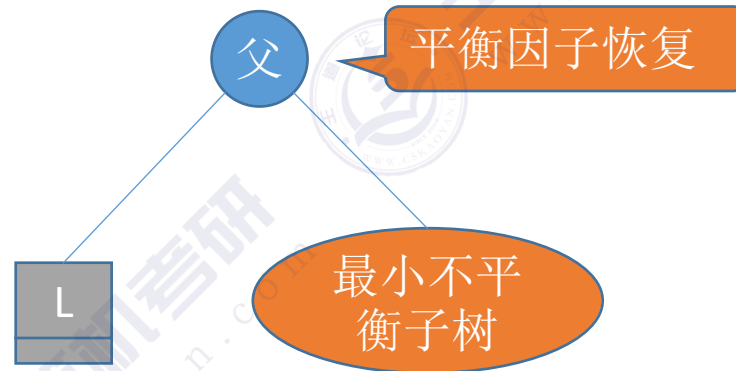
## 填个坑

插入操作导致“最小不平衡子树”高度+1，经过调整后高度恢复



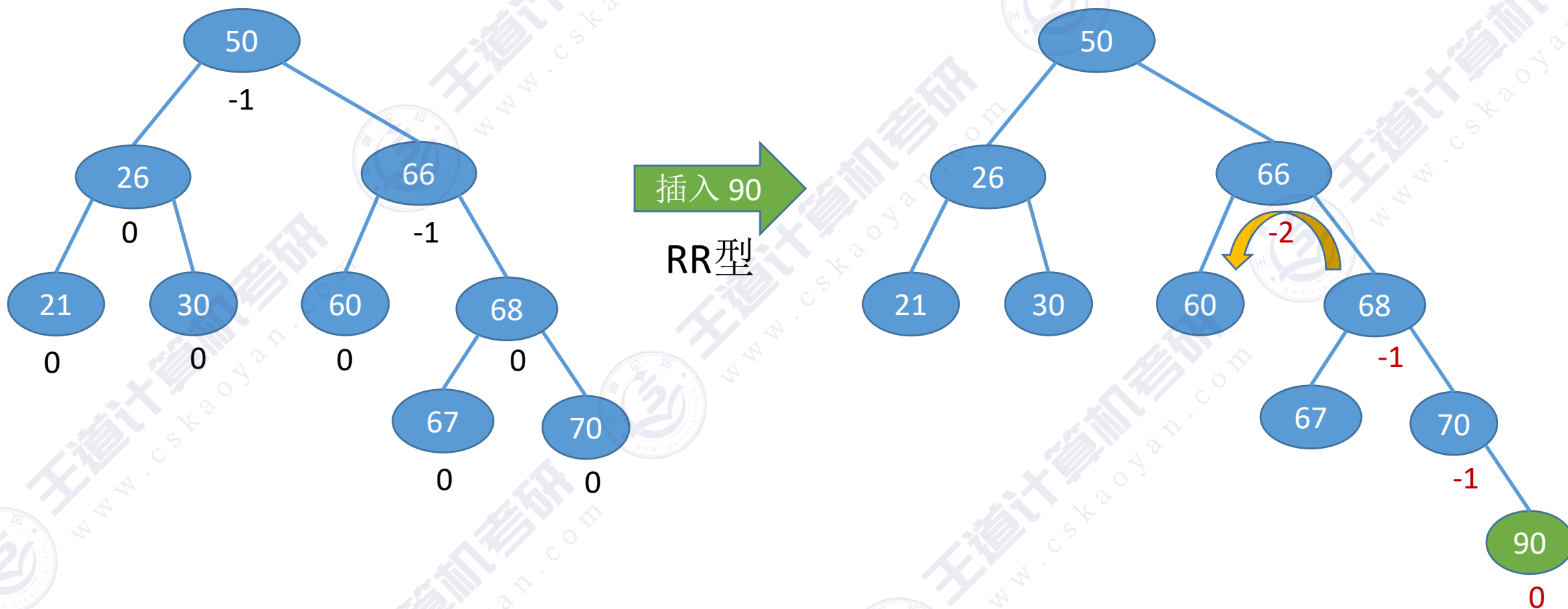
WHY?

在插入操作中，只要将最小不平衡子树调整平衡，则其他祖先结点都会恢复平衡

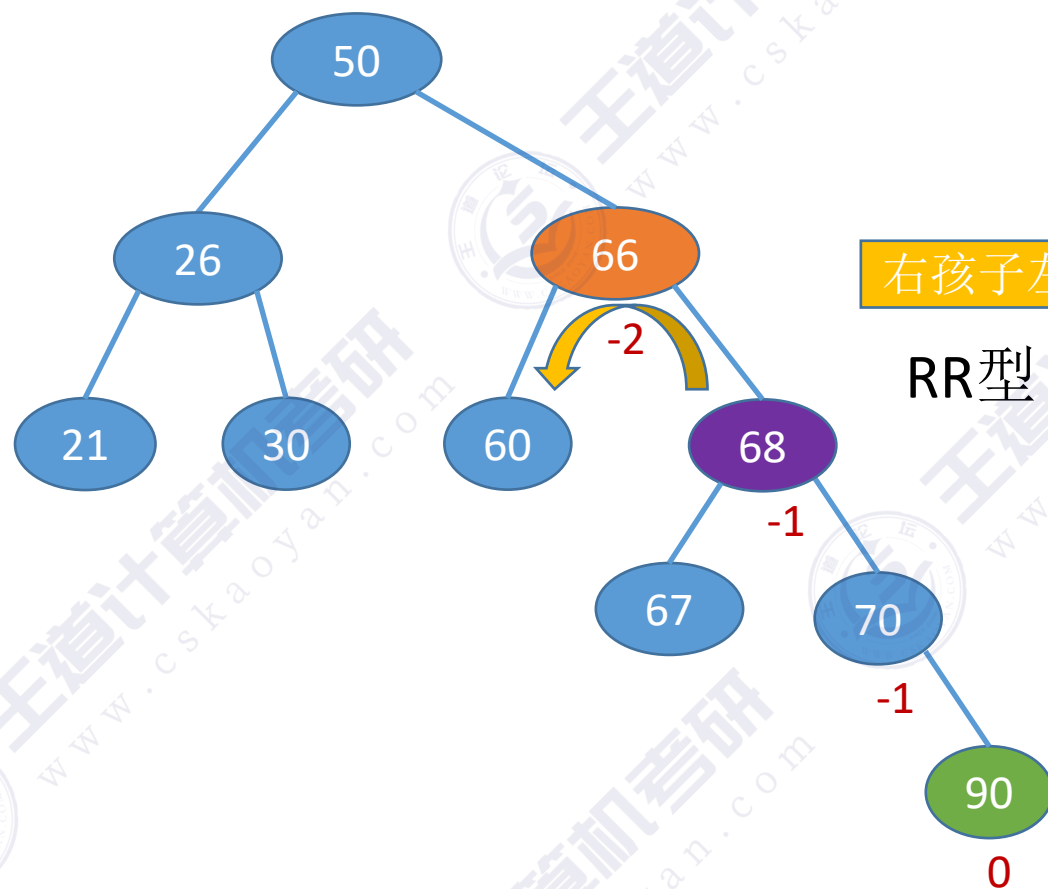




## 练习

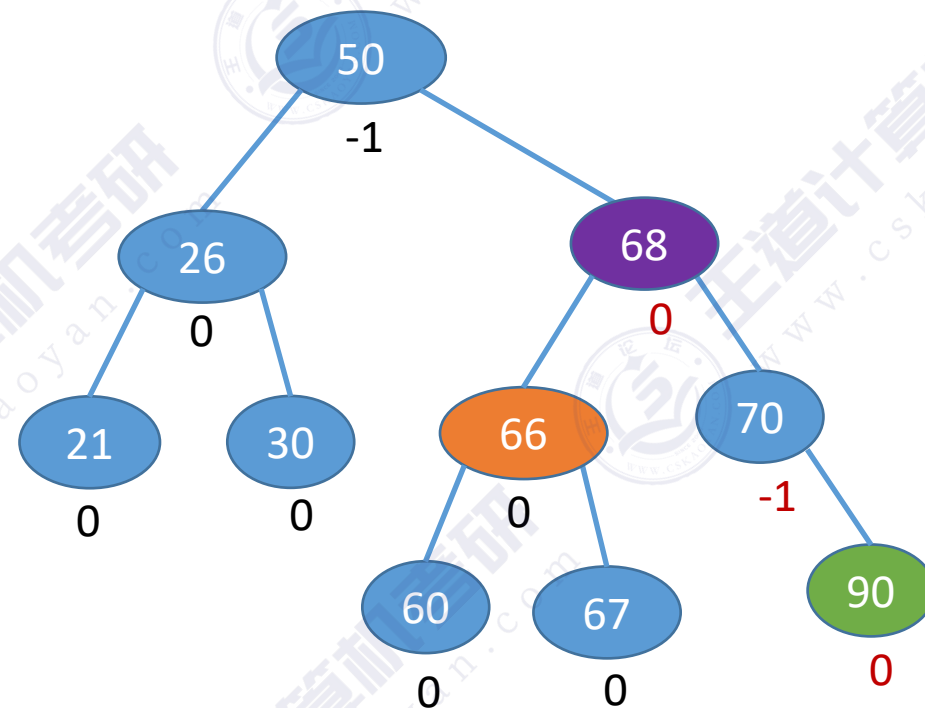


## 练习



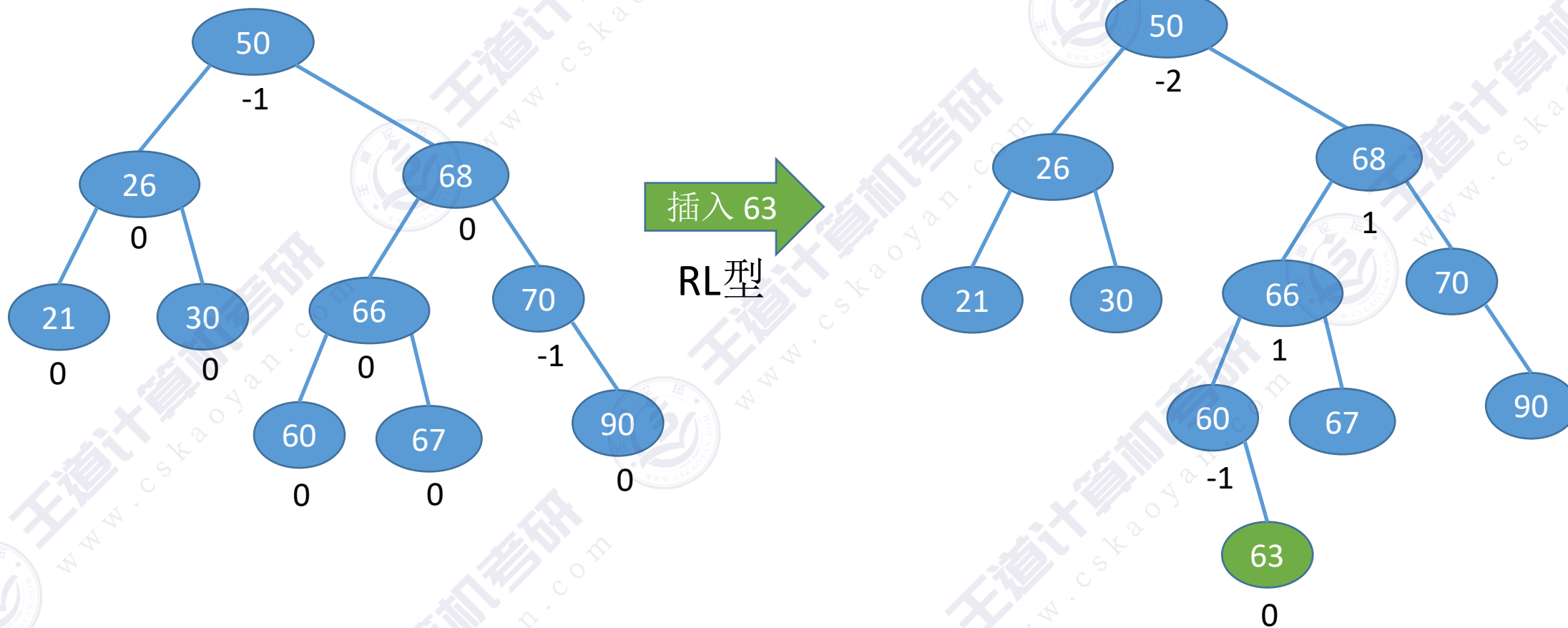
右孩子左旋

RR型

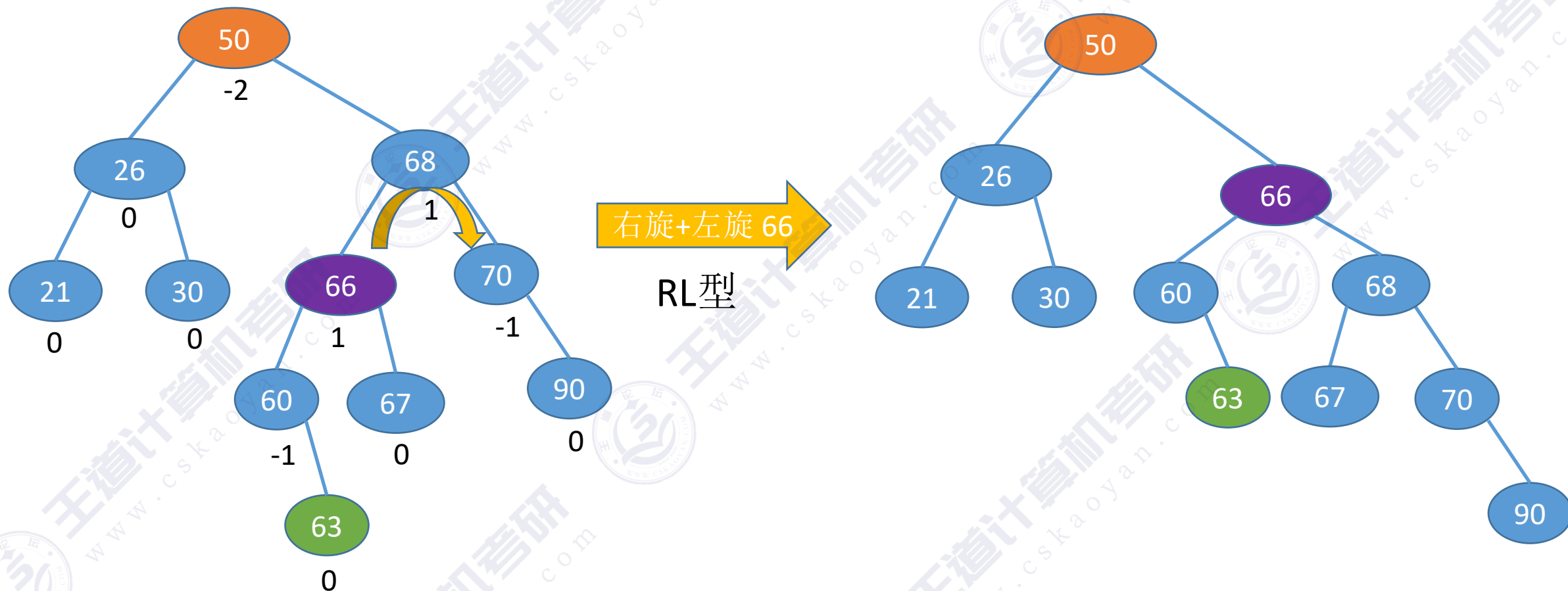


注意检查：是否符合 **左<根<右**

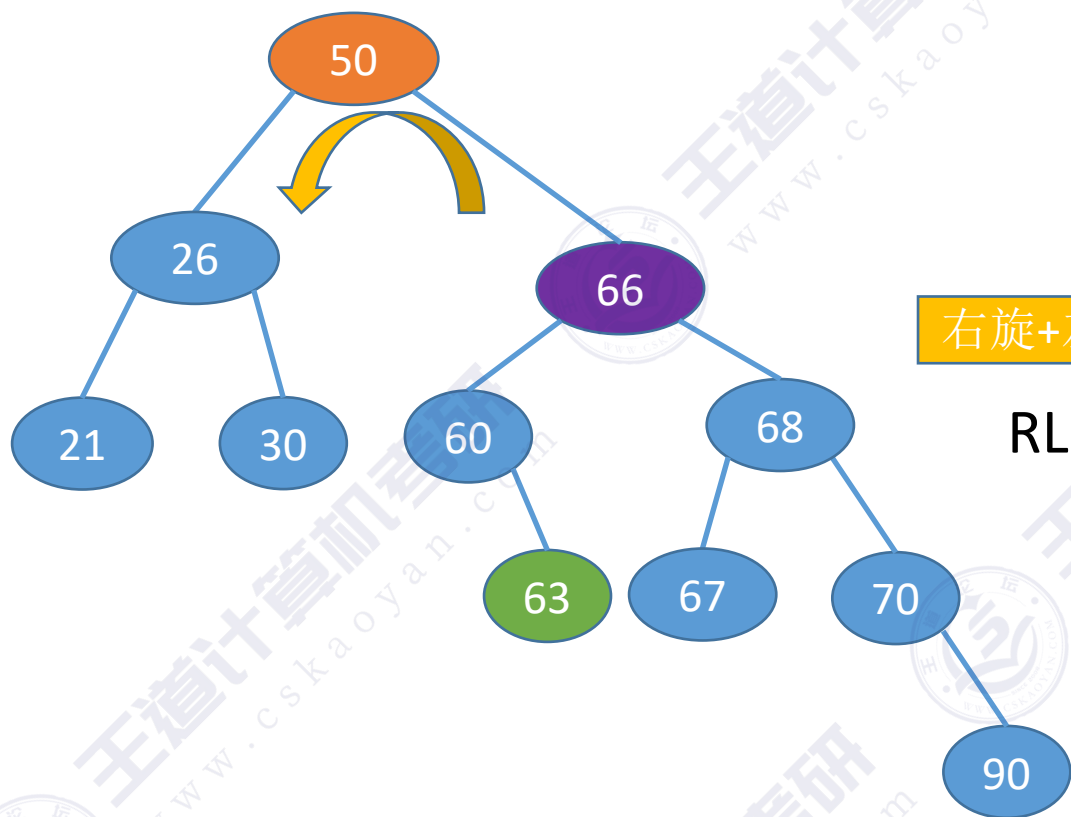
## 练习



## 练习

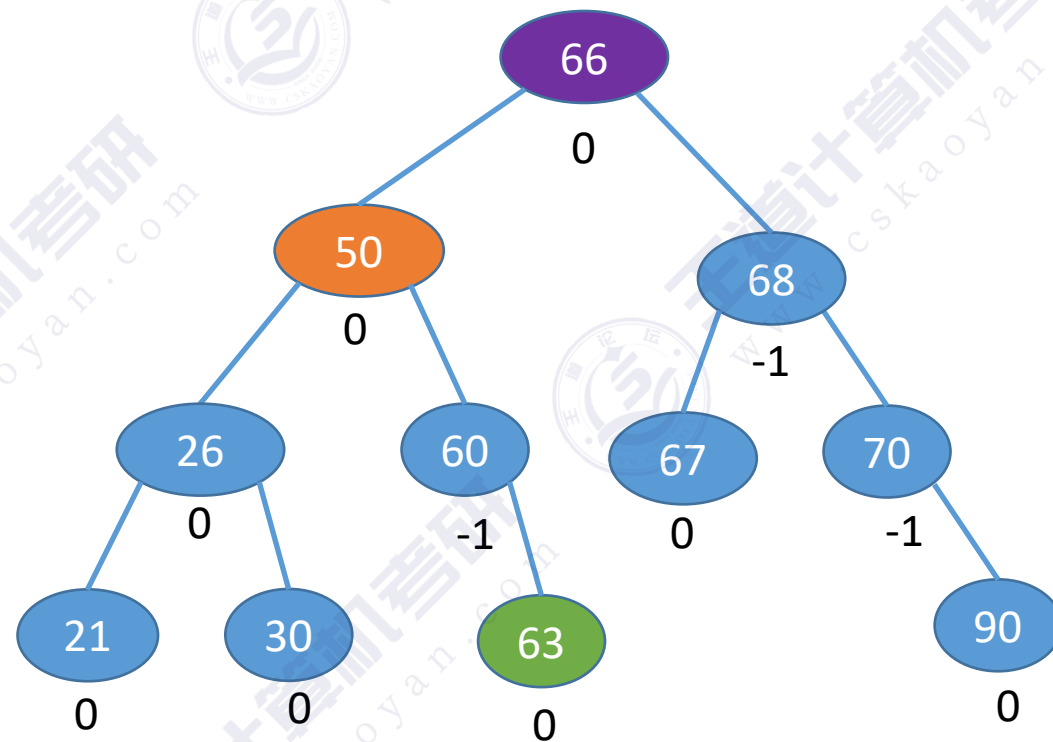


## 练习



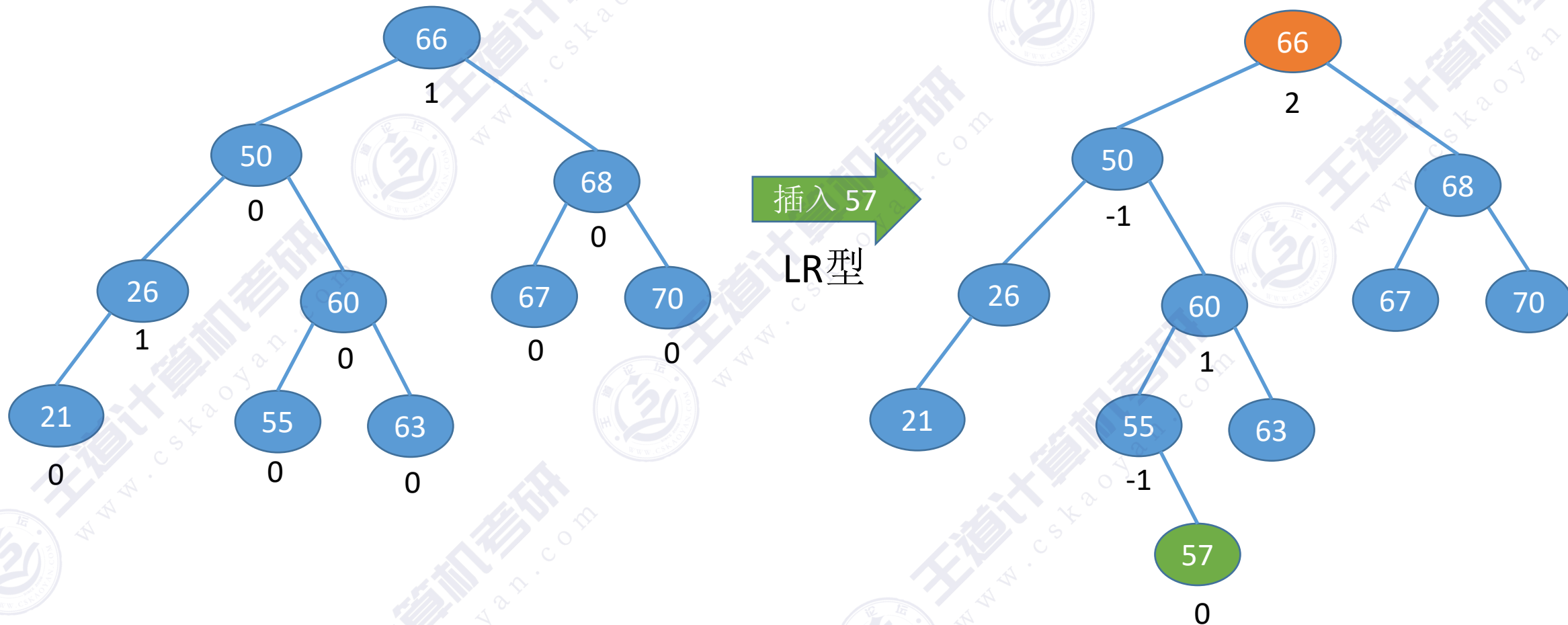
右旋+左旋 66

RL型



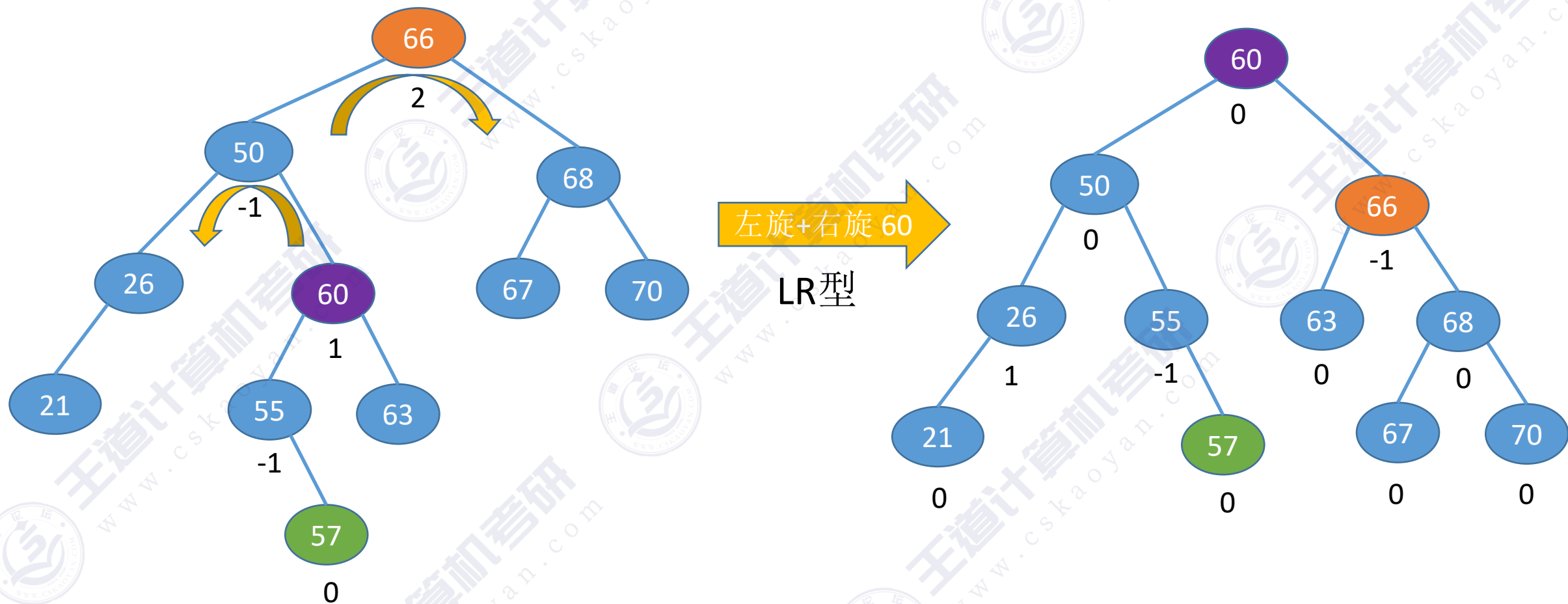
注意检查：是否符合 **左<根<右**

## 练习





## 练习



## 查找效率分析

若树高为 $h$ ，则最坏情况下，查找一个关键字最多需要对比 $h$ 次，即查找操作的时间复杂度不可能超过 $O(h)$

平衡二叉树——树上任一结点的左子树和右子树的高度之差不超过1。

假设以 $n_h$ 表示深度为 $h$ 的平衡树中含有的最少结点数。

则有 $n_0 = 0, n_1 = 1, n_2 = 2$ ，并且有 $n_h = n_{h-1} + n_{h-2} + 1$

可以证明含有 $n$ 个结点的平衡二叉树的最大深度为 $O(\log_2 n)$ ，平衡二叉树的平均查找长度为 $O(\log_2 n)$

# 查找效率分析

《An algorithm for the organization of information》——G.M. Adelson-Velsky 和 E.M. Landis ,1962



Figure 1



Figure 2

The recording algorithm is such that at each moment, the reference board is an admissible tree.

**Lemma 1.** *Let the number of cells of the admissible tree be equal to  $N$ . Then the maximum length of the branch is not greater than  $(3/2) \log_2 (N + 1)$ .*

**Proof.** Let us denote by  $N_n$  the minimum number of cells in the admissible tree when the given maximum length of the branch is  $n$ . Then it can be easily proven (see Figure 2) that  $N_n = N_{n-1} + N_{n-2} + 1$ .

When we solve this equation in finite remainders, we get

$$N_n = \left(1 + \frac{2}{\sqrt{5}}\right) \left(\frac{1 + \sqrt{5}}{2}\right)^n + \left(1 - \frac{2}{\sqrt{5}}\right) \left(\frac{1 - \sqrt{5}}{2}\right)^n - 1.$$

Whence

$$n < \log_{1+\sqrt{5}} (N + 1) < \frac{3}{2} \log_2 (N + 1),$$

关注公众号【研途小时】获取后续课程完整更新！



# 知识回顾与重要考点



实现 f 向右下旋转, p 向右上旋转:  
其中 f 是爹, p 为左孩子, gf 为 f 他爹

- ①  $f \rightarrow lchild = p \rightarrow rchild;$
- ②  $p \rightarrow rchild = f;$
- ③  $gf \rightarrow lchild/rchild = p;$



实现 f 向左下旋转, p 向左上旋转:

- ①  $f \rightarrow rchild = p \rightarrow lchild;$
- ②  $p \rightarrow lchild = f;$
- ③  $gf \rightarrow lchild/rchild = p;$

## 平衡二叉树

定义

树上任一结点的左子树和右子树的高度之差不超过1

结点的平衡因子=左子树高-右子树高

插入操作

和二叉排序树一样, 找合适的位置插入

新插入的结点可能导致其祖先们平衡因子改变, 导致失衡

调整“不平衡”

找到最小不平衡子树进行调整, 记最小不平衡子树的根为A

LL 在A的左孩子的左子树插入导致A不平衡, 将A的左孩子右上旋

RR 在A的右孩子的右子树插入导致A不平衡, 将A的右孩子左上旋

LR 在A的左孩子的右子树插入导致A不平衡, 将A的左孩子的右孩子 先左上旋再右上旋

RL 在A的右孩子的左子树插入导致A不平衡, 将A的右孩子的左孩子 先右上旋再左上旋

查找效率分析

考点: 高为h的平衡二叉树最少有几个结点——递推求解

平衡二叉树最大深度为  $O(\log n)$ , 平均查找长度/查找的时间复杂度为  $O(\log n)$